Towards Informatic Analysis of Syslogs

Jon Stearley
Sandia National Laboratories
jrstear@sandia.gov

Abstract

The complexity and cost of isolating the root cause of system problems in large parallel computers generally scales with the size of the system. Syslog messages provide a primary source of system feedback, but manual review is tedious and error prone. Informatic analysis can be used to detect subtle anomalies in the syslog message stream, thereby increasing the availability of the overall system. In this paper I describe the novel use of the bioinformatic-inspired Teiresias algorithm to automatically classify syslog messages, and compare it to an existing log analysis tool (SLCT). I then describe the use of occurrence statistics to group time-correlated messages, and present a simple graphical user interface for viewing analysis results. Finally, example analyses of syslogs from three independent clusters are presented.

1. Syslog Background and Related Work

All event logs have a timestamp of the event and some log message. In some log domains the vocabulary of terms (or phrases) which can exist in the message are well-defined and limited (I.e. SNMP logs, phone network logs, etc), but syslog vocabulary is notoriously unlimited. The syslog RFC [1] defines a syslog message to contain a timestamp, a string identifying the source of the message (the device generating the message), and a free-format 1024-byte ASCII description of the event (identified as the "MSG" portion of the message). This information triple is what is referred to as "message" throughout this document. Due to its tenure and flexibility, syslog is the most widely available event log protocol. Syslog messages are commonly generated by many devices for many reasons, ranging from hardware faults to normal user actions (login, logout, etc) - resulting in an untenable number of lines of text for review. This glut of data often results in the useful information therein being very inefficiently gleaned, or going completely unused. Furthermore, syslog verbosity levels can be increased in order to provide more information (for example, more advance warning of problems), but this is not often done because the amount and content variance of even terse log verbosity levels can be overwhelming. Steinder and Sethi provide an excellent overview of the approaches to the ongoing need for event log analysis [2], with the specific goal of fault localization in complex networks. Figure 1 provides some sample syslog messages which are used in examples later in this document.

Expert systems [3] comprise the most common log analysis approach, in which a human expert enumerates a set of rules, consisting of regular expressions and responses to take when matching messages are encountered (for example: ignore as benign, alert via email or pager, etc). Swatch [4], SEC [5] , and logsurfer [6] are popular implementations of this approach. Prewett describes using logsurfer specifically for monitoring Linux cluster syslogs [7] - using 1221 hand-written rules! The time and rigor required to generate a sufficiently rich rule set is prohibitive for many sites. If insufficient effort is invested, the resulting false-positive and false-negative rates generally kill off the effort completely. An example false-negative is not receiving an alert for a message deserving an alert, but is ignored due an imprecise "ignore" rule. An example false-positive is to receive alerts for benign messages which match an imprecise "alert" rule. The difficulty of writing and maintaining regular expressions for monitoring is proportional to the number of types of messages present, and the rate at which this set of message types change (for example, additions of new devices or software, or changes in user behavior). Sandia Laboratories Cplant and Institutional Computing Clusters also use logsurfer - but with only 519 lines of rules.

Visualization is another popular approach to log analysis. Takada has produced interesting visualizations of time statistics in 2-D plots [8] and source relationships in 3-D images [9]. Hoagland has explored visualization of audit logs as directed graphs [10]. Girardin [11] has produced spring, self organizing map, and parallel coordinates visualizations of event logs, and Couch has presented tools for visualizing as time series, and various 2-D plots towards anomaly and correlation discovery [12]. While these are very interesting representations of log data, no similarity metric or presentation type is widely accepted as an effective answer to the problem of analyzing logs. In arguably the most novel presentation approach, Gilfix auralizes event sequences [13], complete with sound-scape themes including “wetlands” and “jungle”! While this presentation is extremely creative and effective, it may not result in a widely acceptable long-term work environment, and provides no mechanism to understand exactly what is anomalous in a sequence of messages (“did you hear that?” “hear what?”).

Simple Network Management Protocol (SNMP [14]) generates event logs which generally have very limited and well-documented vocabulary[1] compared to syslogs (by good design!). Most analysis approaches focus on SNMP logs. Hellerstein et al have excellent results on discovering and modeling time and sequence correlation in SNMP logs, including some useful visualization[15, 10]. Using SNMP logs avoids the complexities associated with large-vocabulary free-format messages, but their pattern discovery work is directly applicable to syslogs once message types have been determined. Both log streams are generated over large and complex networks, with correlations among messages from different sources. By contrast, this document focuses on strategies to automatically generate optimally-precise sets of message types.

In summary, the existing and increasing number of publications, tools, and commercial services[2] indicate that event log analysis is a well-studied but open problem.

2. Analyst Thought Process

With the specific goal of increasing supercomputer RAS (reliability, availability, and serviceability), we intend to produce a machine-learning analysis system which enables content-novice analysts to efficiently understand evolving trends, identify anomalies, and investigate cause-effect hypotheses in large multiple-source event log sets. With this in mind (and the assumption that malfunctions and/or misuse should not be not the site norm), we have sought to capture the thought process of log analysts, rather than their log-specific expert knowledge. The overarching question is simply, *“what message stream aspects are strongly correlated with system malfunctions and/or misuse?”* Defining “normal” is of course the elusive key to any anomaly-detection strategy - the following are more specific restatements of the above question, whose quantitative answers we believe should form a sufficient “normal” with which to approach our goal.

Q1 *What message content and occurrence rate is normal?* Which messages are always exactly the same (for example the first two lines in Figure 1)? Which messages are always the same except for certain terms (for example,which numbers in the dns message in Figure 1 normally change over time, and which ones do not)? Does each message type have an associated “normal” occurrence rate (hourly, weekly, bursty when users start work in the morning, etc.)?

Q2 *What message groups are normal?* What sequence of messages indicates a single event? Which messages usually occur proximate in time to each other, and over what time span? Are there messages from multiple sources which form reliable sequences?

Q3 *Can devices be classified by their output message stream?* Are there sets of hosts which produce very similar message streams (for example all compute nodes verses service nodes in a cluster, or hosts which tend to have memory problems, and those which do not)? If so how frequently and why do host classifications change?

Q4 *Can users or applications be classified by their resulting message stream?* Are there distinctive characteristics about the message streams generated while hosts are used by certain users, or executing certain applications?

Q5 *Are device-to-device and/or job-to-job log stream similarities sufficient to identify hardware or software failures?*

This document only addresses the first two questions, the others are topics of ongoing work but are included here for review and discussion.

3. The Sisyphus Toolkit

Log analysis is commonly considered a never-ending curse due to the tedium involved and elusiveness of a successful automated strategy - sharing these views I have named the log analysis toolkit Sisyphus[3]. It does not aim to completely automate the process, but rather provides wedges and levers to speed and ease the task of rolling logs up the hill of system monitoring and debugging. The toolkit is a collection of third-party and original components which currently provides three distinct capabilities: automated generation of message types, automated grouping of time-correlated messages, and interactive review of these results.

[1] “port 5 up”, “port 5 down”, “port 5 traffic threshold exceeded” are example SNMP messages

[2] see http://www.counterpane.com/log-analysis.html and http://www.loganalysis.com for extensive listings of available log analysis tools and services

[3] In Greek mythology, Sisyphus was given the eternal punishment of pushing a large rock up a hill, only to have it always roll back down

```
Nov 25 17:54:03 n-0.t-37 in.rshd[570]:  connect from 192.168.254.3
Nov 25 17:54:48 n-0.t-37 in.rshd[580]:  connect from 192.168.254.3
Dec 20 05:41:18 dns named[285]:  XSTATS 1008852078 1007338854 RR=363555 RNXD=30161 RFwdR=220973 RDupR=775 RFail=1354
RFErr=0 RErr=226 RAXFR=76 RLame=9094 ROpts=0 SSysQ=80234 SAns=2650384 SFwdQ=215128 SDupQ=51336 SErr=0 RQ=2689813
RIQ=5 RFwdQ=215128 RDupQ=4507 RTCP=18234 SFwdR=220973 SFail=13 SFErr=0 SNaAns=444744 SNXD=136333 RUQ=0 RURQ=0 RUXFR=0
RUUpd=5994
Feb 22 07:01:54 l105 CROND[6568]:  (root) CMD (run-parts /etc/cron.hourly)
Feb 22 07:02:11 l227 CROND[23593]:  (root) CMD (run-parts /etc/cron.daily)
Feb 22 07:03:46 l227 CROND[23593]:  (root) CMD (run-parts /etc/cron.weekly)
Feb 22 07:05:15 l102 CROND[28698]:  (jack) CMD (/bin/rm ~/gram_job_mgr_*.log)
```

Figure 1. Sample syslog lines

3.1. Using Teiresias for Automated Message Typing

Teiresias[4] is a pattern discovery algorithm originally developed for bioinformatics at IBM which has received considerable attention [16, 17, 18] and has been applied outside its domain, including for security anomaly detection [19]. It discovers all patterns (called "motifs") in categorical (non-numeric) data of at least a user-given specificity and support. More explicitly, given a set of strings X composed of characters C, Teiresias finds all motifs M (composed of characters C and a don't-care wildcard "$\star$") having at least a specificity of L/W (where L is the number of fixed characters from C, and W is the total width of the motif including wildcards), which occur at least K times in X. It first hashes all motifs of length W appearing in the data, then prunes those not meeting the minimum support criteria K, and then conducts a "convolution" phase where all remaining motifs are glued together in order to yield the longest motifs existing in the data (leveraging the property known as "downward closure" [20]). Output motifs are carefully sorted by decreasing specificity and maximality. For example, suppose motif m_i and m_j match exactly except that m_j is shorter (less specific). If they occur the same number of times then m_j is discarded and only the "maximal" motif m_i is output. If however m_j has higher support, it does present additional information about the data, and is thus included in the output, but only after m_i. See the above references for more details on how this is accomplished (in worst-case exponential time, but most datasets yield results in polynomial time). These features, coupled with its downloadable-for-research license, convinced us to explore its applicability to event log analysis.

Teiresias was originally developed to work with only very small vocabulary sets, but was later also released as *teiresias_int,* which is capable of handling a vocabulary of 32-bit integers (deemed sufficient for log analysis). Before logs can be analyzed with *teiresias_int* (which must be downloaded separately), they must be preprocessed by substituting a unique integer for each unique word (and the reverse mapping is performed on the resulting *teiresias_int* output). This is accomplished by a Perl script named *dictify* which splits words on whitespace, with the following exceptions:

- Message timestamp and source are omitted - they are deferred to a later stage of examination.
- Words containing an equals (=) sign are split into two, in order to accommodate messages containing words such as "`variable_name=value`" (I.e. the dns line in Figure 1). The pair is reattached upon conversion back into strings.
- Numeric Process IDs (PID) are omitted. It is common for messages to contain "`program_name[PID]:`", where the PID changes from one invocation to the next. If PID were not omitted, the resulting integer given to Teiresias would obscure the common "`program_name`" prefix. I currently consider the benefit of exposing the common `program_name` to Teiresias to outweigh the loss of PID, which is converted to a wildcard upon reverse conversion into words (I.e. "[*]: "). Preserving PID by splitting as is done with `variable_name=value` pairs may be attempted in the future, as it would enable the detection of extra or missing processes during the startup of identically-configured nodes. For example, the "PCT" process is always PID 430 on Cplant compute nodes (not functionally necessary, but is certainly "normal").

The word-integer mapping is saved in file *X.dict*, and the now integer-only syslog *X.dat* is passed to *teiresias_int* which finds all L,W,K motifs (resulting in *X.thr*). The *classify* script then compares each message against this output, noting only the first match for each - effectively selecting only the maximal motifs from the total set. It then performs the multi-line grouping described in Section 3.3. Finally, these results are converted back into strings (*dictify -r*), saved to file *X.rex*, and displayed with *logview* (Section 3.4). Messages not matching any motifs (outliers) are saved to the file *X.nolab* for review, or for iterative processing with different arguments. The entire process is driven by a front-end script called *teirify* which takes *L W K* as arguments (with meanings as described above). Figure 2 shows the complete flow of data through *teirify.*

[4] Teiresias was a Zeus-empowered prophet and sage in Greek mythology. Download or ask teiresias questions at http://cbcsrv.watson.ibm.com/Tspd.html

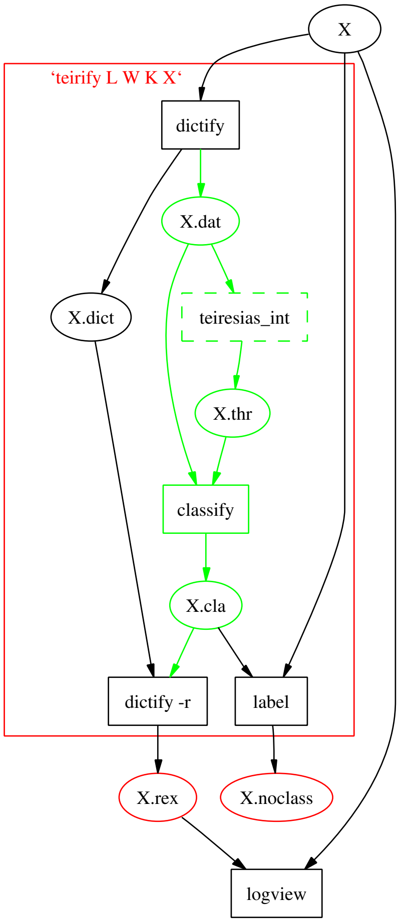

Figure 2. Graph of data flow through the *teirify* program. Ellipses denote files, boxes denote programs, *teiresias_int* is dashed to indicate that it is not included in the Sisyphus toolkit (separate download from IBM).

As a simple example, consider a log file containing the lines in Figure 1, along with an additional 3118 CRON lines (additional 2787 hourly, 232 weekly, and 99 daily messages exactly like the ones in Figure 1). *'teirify 2 3 100'* results in four motifs:

```
CROND[*]:  (root) CMD (run-parts /etc/cron.hourly)
CROND[*]:  (root) CMD (run-parts /etc/cron.weekly)
CROND[*]:  (root) CMD (run-parts /etc/cron.daily)
CROND[*]:     *    CMD
```

With support of 2787, 232, 99, and 1 - highlighting the last as a rare event (but similar to common events, since it shares two words with all the other CRON messages). The other lines are appropriately assigned to a NOCLASS group, as they are unlike all other lines in the file. In effect, *teirify* places the messages into three categories of increasing anomaly:

- messages having a support of at least *K* (frequent),
- messages having a support less than *K* (infrequent) but are similar in content to frequent messages,
- messages which are both infrequent and anomalous in content.

This categorization enables extremely efficient review of the log file. It is possible to start with a large value of *K*, and then iterate on *X.nolab* using successively smaller values, also decreasing the *L/W* ratio. I have found this largely unnecessary for log data, and tend to always use *'teirify 2 3 10'*. For example, the above results do not change for K ranging from 2 to 231, or for a lower ratio of *L/W*. Further examples are provided in Section 4.

It is not practical to use Teiresias to examine partial words (for example, convert characters instead of words into integers and then pass through *teiresias_int*) because it does have support for variable-sized gaps in motifs. Whereas this could be attempted, the *L/W* ratio required to generate useful results would likely prove computationally prohibitive. This also makes it impractical for use in discovering sequences of messages, because message sequences interleave differently according to system activity, which is extremely variable (true for a single host, let alone a log set from multiple sources).

The sole difficulty I have encountered with Teiresias is very significant however: the vocabulary size of most syslogs requires very large amounts of system memory - I have only rarely been able to process more than 10,000 lines at a time using 1.5GB of memory. I believe this is due to the fact that *all* motifs of *L/W* specificity are enumerated during the first phase. This implies that *all* words are frequent, which is not the case for syslogs [21]. It is very likely that if Teiresias were modified to first determine which words have a support of at least support *K* before motif enumeration (as does SLCT, see next section), the bottleneck would be avoided. This bottleneck can be currently mitigated by passing a small log file through *teirify*, using the results to filter out all matching messages from a larger file, and then passing the resulting outliers back through the process - iterating until a sufficiently large time range has been analyzed.

Despite its limitations to small data sets, Teiresias' input-parameter insensitivity for log data, automatic message typing, and anomaly categorization show it to be a very useful and effective pattern discovery engine for automatic typing of syslog messages.

3.2. Using Simple Logfile Clustering Tool (SLCT) for Automated Message Typing

Risto Vaarandi's SLCT[5] uses an algorithm specifically designed to detect word clusters in log messages [21]. It makes three passes through the data to accomplish this objective. A hash counting all words and their position in the line is generated on the first pass through the data ("the dog ran" is be hashed into three keys: "1_the", "2_dog", "3_ran"). Words having a support less than *s* are then pruned from the hash, and a new hash of message word clusters is generated during a second pass through the data (the messages "the dog ran" and "the deer ran" would generate a key of "1_the 2_* 3_ran" for *s=2* - the second word is the wildcard "*" since dog and deer only appear once). An optional third pass can be performed in which wildcard positions are refined with constant heads or tails if possible (in our example, "2_*" becomes "2_d*" because both dog and deer begin with d). The resulting word cluster and their support is output, and any lines not matching any word cluster are saved to a separate file for review ("outlier" lines). SLCT executes faster than Teiresias, its results are very similar, and it does not become memory-bottlenecked as Teiresias does (enabling it to process very large log sets). SLCT provides many command line arguments, enabling very flexible behavior, but also requires some careful thought in order to produce good results. For example, *slct -s 10 -b 16 -f '[^[:space:]]+ (.+)' -t '$1' -r -j logfile* tends to produce good results for standard syslog files (specifies a support threshold of 10, omit first 16 bytes and next word from consideration (timestamp and message source), refine and join clusters). The addition of options to specify common log formats (for example *-syslog* might result in *-b -f -t* options similar to the above) would significantly improve its usability.

There are two technical issues to overcome with SLCT. The first is simply that no mechanism is provided to review which raw lines match which word cluster. This is a significant impediment in understanding its results (particularly on logs resulting in diverse word clusters), and contributes to the difficulty of identifying the optimal set of command line arguments (minor changes to input arguments can have drastic effects). To address this I am working to enable its results to be reviewed with *logview* (see Section 3.4). The second issue is related to SLCT's optional join (*-j*) option, which approximates Teiresias' ability to categorize message types. Again consider the example CRON logfile. *'slct -s 100 -r'* results in two word clusters:

```
CROND[*]:  (root) CMD (run-parts /etc/cron.hourly)
CROND[*]:  (root) CMD (run-parts /etc/cron.weekly)
```

with correct support of 2787 and 232 respectively, and saves the remaining 103 lines in an outliers file. Adding the *-j* option results in two additional word clusters:

```
CROND[*]:    *    CMD    *       *
CROND[*]:  (root) CMD (run-parts *
```

with supports of 3119 and 99 (saving only 3 messages as outliers). This does not clearly present additional information about the log, failing to effectively highlight the single message about user `jack`, or present the daily messages as a separate group. If the support threshold is decreased below 100 (*-s 99* for example), the latter word cluster becomes:

```
CROND[*]:  (root) CMD (run-parts /etc/cron.daily)
```

with support of 99, but user `jack` is still lost in the most general CRON word cluster. While this CRON example is intentionally simple, it illustrates a difficulty with using *-j* which tends to quickly obscure subtle anomalies in real log files, especially when combined with trial-and-error attempts to identify the optimal value for support threshold (*-s*).

There is preliminary support via a front-end script called *slctify* which passes a log file through SLCT and displays results with *logview*, this does not support the *-j* option at time of this writing however. Given SLCT's many strengths and minor shortcomings, I expect it to replace Teiresias as the preferred message typing engine in Sisyphus.

3.3. Automatic grouping of time-correlated messages

Having generated single-line message types, attention is turned to grouping messages together which are correlated in time. The median and standard deviation of inter-arrival periods for all messages in each message type is calculated by the program *classify*, and the message types are then grouped according to these statistics. For example if

[5] SLCT is GPL-licensed and available at http://kodu.neti.ee/~risto/slct

message type m_i matches line numbers 1, 7, 22, and 23 in file *X*, and the timestamps for those lines are (in seconds) 60, 120, 180, 240, the median inter-arrival period is 60, with a standard deviation of 0. This simple example is a minute-periodic message type. Such (cron) messages are common at various periodicity, and are presented in their period groups regardless of whether they happen proximate to each other in time. The median is used as it is somewhat more robust to outliers than the mean, such as when "normal" periodic messages are not produced while a node is turned off, which can drastically skew the inter-arrival mean. Ma [22] describes using a chi-squared test of the inter-arrival period histograms against a random-arrival (Poisson) distribution to detect "partially-periodic" events. This is certainly a more robust (and computationally expensive) approach, but I have found the simpler statistics sufficient for a first-order grouping of time-correlated messages. They are useful even for non-periodic messages, because different message types with similar support, inter arrival mean, and inter-arrival standard deviation are in fact time correlated and are thus grouped together. For instance consider one thousand similarly-configured computers booting - they will all emit a similar sequence of messages over a similarly short time duration, regardless of node boot order or concurrency. Each message in the boot sequence will be typed as described in the previous sections, and the set will be grouped together using their occurrence statistics. This enables the entire set to be presented as a single unit unit of one thousand "boot events" - a huge compression of the syslog record, while still highlighting anomalous deviations in the boot events (for example, the rare change in a hardware-check startup message, the occurrence of a message which is "normally" not present during boots, a change in the amount of memory detected (likely indicating a failed DIMM), etc). This example is revisited in Section 4. Using complete inter-arrival histograms for the grouping certainly would be more robust, and I plan to implement this at some point in the future. Message type occurrence statistics are included in the resulting *X.rex* file for review.

3.4. Interactive Review

An interactive Perl-Tk tool named *logview* is provided in the Sisyphus toolkit, which enables efficient review of the above results versus the original log file. A snapshot of the application is not included in this document, but it is simple to describe. The contents of *X.rex* are presented in an upper window, and upon selection via the mouse the corresponding raw messages appear in a lower window. Multiple types can be selected via shift and control clicks, enabling comparison of message types, such as comparing frequent messages to infrequent-but-similar messages. Once a questionable message has been identified, one of the first questions a reviewer often has is, "at what other times and from what other sources does this message appear?" *logview* presents the answers to these questions with a single click, showing all the messages which match the selected message type. It is useful to display the messages in type blocks for this review. If the reviewer wishes to see the time-interleaving of multiple message types, an option is provided to display messages in their original order. *logview* is intentionally simple, providing an extremely intuitive and efficient means of reviewing system logs using the results of the analyses described above.

4. Example Usage

Whereas it is impossible to demonstrate the efficiency these interactive tools provide, this section presents some brief examples of using Sisyphus to review logs from three separate computer clusters as an attempt to further illustrate its usefulness.

4.1. CPlant - Sandia National Laboratories Computational Plant

There are multiple CPlant Linux clusters at Sandia National Laboratories, the largest of which consists of over 1800 nodes. All but 24 nodes are disk-less, and every node in the system generates syslog messages which are transmitted and saved centrally using the excellent syslog-ng[6] program. The system generates an average of 435,000 syslog lines per day, and as mentioned previously logsurfer is used to monitor the production message stream. Cplant however serves as both an operating system research and production computation plant, and the message stream generated during each role is distinctive. Considerable effort has been made to minimize the total time to reliably boot all nodes in the cluster [7], and identifying minor anomalies in the resulting bolus of messages is a challenging task.

An excerpt of running *'teirify 2 3 10'* on a 2840-line file from a single Cplant rack produces 97 message types (excerpt shown in Figure 3, matching all but 27 messages. Immediately one notices a group of message types having a support of 32, and a smaller group having a support of 33 - both groups having nearly identical inter-arrival statistics (except for stddev). By selecting messages in each group one can quickly see that whereas all the nodes in the set booted Linux, one node did not start the Cplant runtime software (an intentional configuration in this case, but demonstrates detection of an easily-missed variation in the message stream). Comparing messages in

[6] syslog-ng available at http://www.balabit.com/products/syslog_ng/

[7] Using the Cluster Integration Toolkit, all 1800 disk-less nodes of Cplant can be routinely booted in under ten minutes as described by Laros and Ward [23]

```
### Class Definitions ###
# label k median stddev motif
L0     27    0    0     NOCLASS
L137   64    0    1     rte:  succeeded
L47    33    1   80     RAM disk driver initialized:  16 RAM disks of 32768K size
L48    33    1  180     eth0:  Digital DS21143 Tulip rev 65 at 0x8000, * IRQ 29.
L53    33    1  180     eth1:  Digital DS21143 Tulip rev 65 at 0x8800, * IRQ 30.
L95    32    1    3     HWRPB cycle frequency (462962962) seems inaccurate - using the measured value of * Hz
L122   32    1    3     if=eth0, addr=* mask=255.255.255.0, gw=255.255.255.255,
L125   32    1    3     bootserver=192.168.37.2, rootserver=192.168.37.2, rootpath=/cluster/machine/diskless/rh-6.2-alpha/image
L146   32    1    1     rte-init:  /cplant/init.d/enfs_client running:  mount_nfs start
L144   32    0    1     rte-init:  1816 routes read (1816 valid).  Max 15, avg 0.003855
L80    31    0    2     init.c:118 Using Alpha PCI_OFFSET_TSUNAMI (0xfffffd0000000000)
L118   28    1  196     Sending DHCP requests .., OK
L136   28    1    3     Memory:  1033592k available
L61    26    1    1     rte-init:  Found a LANai type 7.2 with 2097152 bytes (2048kB) of memory unit 0
L62     6    3    6     rte-init:  Found a LANai type   * with 2097152 bytes (2048kB) of memory unit 0
L119    4    6    2     Sending DHCP requests * OK
L177    4    2   27     Memory:  * available
L131    3    0    0     startup succeeded
L86     1    0    0     Looking up port of RPC 100005/1 on
L94     1    0    0     IP-Config:  * Got DHCP answer from
L120    1    0    0     Sending DHCP
L123    1    0    0     if=* addr=* mask=255.255.255.0, gw=255.255.255.255,
L126    1    0    0     bootserver=* rootserver=* rootpath=/cluster/machine/diskless/rh-6.2-alpha/image
L134    1    0    0     Command line:  ip=* root=/dev/nfs
### 97 of 178 motifs used ###
### Class Membership List ###
L0 9 10 45 50 55 56 57 58 59 60 61 62 152 153 154 155 156 157 158 159 160 161 162 163 164 165 166
L1 1 2 3 4 5 6 7 8 21 22 23 24 25 26 27 28 29 30 31 32 33 34 35 36 37 38 39 40 41 42 43 44 46 47 48 49 51 52 53 54
...
### 2813 of 2840 (99%) lines matched (not in NOCLASS) ###
## 359 of 718 dictionary words used in above classes ###
```

Figure 3. sample *X.rex* excerpts

the two sets reveals that this node booted separately, and then the others booted concurrently, accounting for the different inter-arrival standard deviation. The existence of similar message types L136 and L177 quickly highlight the fact that four nodes have a differing amount of memory than the others. Similarly it can be seen from message types L61 and L62 that 26 nodes contain a LANai type 7.2 network card, but 6 others contain a different LANai version (two clicks in *logview* shows exactly which nodes contain the differing version, 9.0 in this case). Very quickly useful information has been gained from the log regarding both software configuration and hardware differences. By selecting message types L118 and L119 it can even be observed that the DHCP messages for four nodes differed from the rest by having "`.?.,`" instead of "`..,`" - indicating slight network contention impacting four hosts which in this case is harmless, but underscores Teiresias' ability to highlight very minor anomalies which easily go unnoticed in manual review. A few clicks answers the question if these nodes are the ones with different LANai cards or less memory - no in both cases, but it is painless and efficient to get at these answers. Selecting the NOCLASS type displays 27 messages, including two which indicate a "`TSUNAMI machine check - correctable ECC error`" on a single node, five pairs over the same time period indicating "`ABORT_LOAD FIRST TRY ... unknown job ID`" and "`nfs: task 64052 can't get a request slot`", and the remainder indicating warning messages pertaining to NFS configuration. These lines are correctly presented as the set deserving further investigation.

SLCT results for this log produced similar results, producing 73 word clusters matching all but 64 outlier (NOCLASS) messages. Differences from Teiresias directly follow the discussion of SLCT in Section 3.2. One additional observation from comparing their results is that Teiresias generated a message type of "`root@admin-0 as root`", matching messages of different actual format but containing that common substring. While the messages are in fact related to a single event (a reliable sequence from `pam_rhosts_auth`, `PAM_pwdb`, and `in.rshd` processes), the vague message type violates the intention of generating one message type being produced for each actual format type (I.e. one message type per `printf` in C code). SLCT does not produce this cluster due to the fact that the words appear in different positions in the messages. This is deemed another advantage of SLCT over Teiresias, and indicates that word position information is useful.

An initial study of the content novelty rate in Cplant's message stream has also been conducted. The top plot in Figure 4 displays the number of production messages per day (all nodes), and the bottom plot displays the number of word clusters generated by SLCT given various support thresholds (*-s* ranging from 2 to 50). With s=10, 697 word

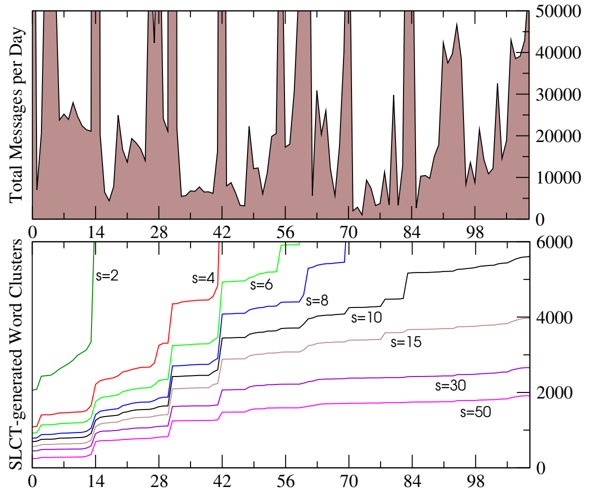

Figure 4. Message and Content Novelty Rate on Cplant

clusters are generated on the first day, but the set grows relatively slowly thereafter, experiencing rapid increases only occasionally. These rapid increases in the number of word clusters reflect bursts of content novelty in the message stream, but they are not *always* correlated with bursts in the raw message stream. For example, even the *s*=2 line in Figure 4 does not rise significantly during the message bursts on days 3, 4, and 5. Similarly, an alarm set for 35,000 messages per day (set median is 19,347) would trigger 29 times, whereas only four significant content novelty bursts are seen with *s*=10. For all values of *s*, the message content novelty rate is approximately constant most of the time - this suggests the possibility of automatically determining a content novelty rate threshold (likely in units of new word clusters per time window) useful towards detecting unusual information bursts in the message stream. This is a topic of ongoing research.

4.2. Liberty - Sandia National Laboratories Institutional Computing Cluster

Liberty is a 256-node dual-XEON cluster at Sandia which also runs Linux, has central syslog collection via syslog-ng, and logsurfer monitoring. Sisyphus was used to speed the process of creating the logsurfer configuration (using both the message types as regular expressions, contexts utilizing the time statistics). The system administrator who performed the task believed Sisyphus had reduced[8] the total time required to establish the logsurfer by roughly 75% - and that a significant portion of that time was spent manually converting from *X.rex* syntax into logsurfer syntax (output to logsurfer syntax is planned for a future revision). The administrator who established the Cplant configuration expressed his disappointment that Sisyphus was not available earlier.

’teirify 2 3 10’ of a 6800-line logfile from a single node shows that 1288 lines are generated exactly every 30 minutes, suggesting it might be worthwhile to tone down CRON messages. 39 messages appear in the NOCLASS type: “`End of File...`”, “`task_check: cannot reply to ...`”, and “`im_eof: Premature end of message`”, the latter indicating scheduler problems on this node. Again the NOCLASS type correctly highlights those messages warranting further attention.

4.3. LosLobos - University of New Mexico High Performance Computing Cluster

‘teirify 2 3 10‘ was run on 7000 lines of syslogs generated by 235 hosts over 12 hours, producing 58 message types matching all but eight messages (NOCLASS). Five of these eight lines indicated that a single host had a different version of a program, and three lines indicated that a port scan had occurred. After careful manual review of the log, these lines were indeed the most interesting/anomalous. Something as simple and minute as a software version mismatch in one node of a parallel cluster can cause numerous problems - the ability to quickly detect such inconsistencies is a valuable debugging tool! Furthermore, automatically highlighting the three lines in 7000 indicating an unauthorized port scan demonstrates value as a security tool as well - anything which is rare in the message stream is flagged as NOCLASS.

slct -s 10 on the same dataset produced 47 word clusters, saving 50 messages as outliers. Differences with Teiresias again match those described in Section 3.2.

As an aside, I have also analyzed UNM’s Computer Science department logs with these tools, and *teirify* does correctly detect which numerical fields in the hourly dns statistics messages (I.e. third line in Figure 1) normally change and which do not.

5. Conclusions and Future Work

Advances resulting from recent international priorities on informatics should be used as widely as possible, including for system event log analysis. The bioinformatic-inspired Teiresias algorithm has been shown to be effective in automatically generating word-granular regular expressions for system event logs. Furthermore its careful sort order provides a near-optimal categorization of messages

[8] Sisyphus-aided initial establishment of logsurfer rules for Liberty took roughly 12 hours total

into frequent, infrequent but content-similar to frequent, and infrequent and content-novel sets. Whereas its memory requirements do not allow it to scale to very large log sets, it provides very useful analysis of logs under 10,000 lines long. SLCT produces similar but less-effective results but does not suffer from prohibitive memory requirements. Whereas Teiresias could potentially be modified to not include infrequent words in candidate motifs (likely to eliminate the memory issues), it also has an algorithmic weakness compared with SLCT in that it does not use word position information (shown above to produce erroneous motifs in some cases). These issues coupled with SLCT's open-source availability (and a friendly and helpful author!) make it the message-typing engine of choice, and I am modifying it accordingly. Very large log sets can be automatically decomposed into message types. Once messages are typed, sequences of messages can be automatically grouped using their inter-arrival statistics, providing further consolidation of messages for review or subsequent analysis.

Determining the optimal sort order for reviewing message types is a topic of ongoing work, involving a combination of appearance order, type-to-type similarity, specificity, and support. The process of establishing rich monitoring rule sets could be further streamlined by outputting SEC or logsurfer syntax directly from SLCT (or at least, automatic conversion of output into appropriate syntax). Another topic of ongoing work is the automatic determination of content novelty rate thresholds (likely in units of new message types per unit time) useful in detecting the most anomalous region of an event log.

The Sisyphus toolkit provides a good start towards informatic analysis of syslogs by presenting messages in a very high signal-to-noise manner, enabling efficient log review and generation of monitoring rules - useful towards the detection of component failure, misconfiguration, and misuse.

Acknowledgement Many thanks to Nathan Dauchy, George Davidson, Tim Draelos, Risto Vaarandi, Kevin Boyack, Donna Johnson, and Jerry Smith for the many helpful discussions on this topic and reviews of this document.

Sandia is a multiprogram laboratory operated by Sandia Corporation, a Lockheed Martin Company, for the United States Department of Energy under Contract DE-AC04-94AL85000.

References

[1] C. Lonvick, "The bsd syslog protocol." RFC3164, August 2001.

[2] M. Steinder and A. Sethi, "The present and future of event correlation: A need for end-to-end service fault localization," 2001.

[3] P. Jackson, *Introduction to Expert Systems.* International Computer Science Series, Addison Wesley, 1986.

[4] S. E. Hansen and E. T. Atkins, "Automated system monitoring and notification with swatch," in *USENIX LISA'93 Conference Proceedings*, 1993.

[5] R. Vaarandi, "Sec - a lightweight event correlation tool," in *IEEE IPOM'02 Proceedings*, 2002.

[6] Logsurfer is available at http://www.cert.dfn.de/eng/logsurf/ and Logsurfer+ at http://www.crypt.gen.nz/logsurfer/.

[7] J. Prewett, "Analyzing cluster log files using logsurfer," in *Proceedings of the 4th Annual Conference on Linux Clusters*, 2003.

[8] T. Takada and H. Koide, "Mielog: A highly interactive visual log browser using information visualization and statistical analysis," in *USENIX LISA'02 Conference Proceedings*, 2002.

[9] T. Takada and H. Koide, "Information visualization system for monitoring and auditing computer logs," in *IEEE Conference on Information Visualization*, 2002.

[10] G. Liu, A. Mok, and E. Yang, "Composite events for network event correlation."

[11] L. Girardin and D. Brodbeck, "A visual approach for monitoring logs," in *USENIX LISA'98 Conference Proceedings*, 1998.

[12] A. L. Couch, "Visualizing huge tracefiles with xscal," in *USENIX LISA'96 Conference Proceedings*, 1996.

[13] M. Gilfix and A. L. Couch, "Peep (the network auralizer): Monitoring your network with sound," in *USENIX LISA'00 Conference Proceedings*, 2000.

[14] J. Case, "Simple network management protocol (snmp)." RFC1157, http://www.ietf.org/rfc/rfc1157.txt, May 1990.

[15] J. L. Hellerstein, S. Ma, and C. Perng, "Discovering actionable patterns in event data," *IBM Systems Journal*, vol. 41, no. 3, 2002.

[16] I. Rigoutsos and A. Floratos, "Combinatorial pattern discovery in biological sequences: the teiresias algorithm," *Bioinformatics*, vol. 14, no. 1, 1998.

[17] A. Floratos and I. Rigoutsos, "On the time complexity of the teiresias algorithm." IBM Technical Report RC21161, 1998.

[18] B. Brejova, C. DiMarco, T. Vinar, S. R. Hidalgo, G. Holguin, and C. Patten, "Finding patterns in biological sequences." University of Waterloo, project report for CS798G, Fall 2000.

[19] A. Wespi, M. Dacier, and H. Debar, "An intrusion-detection system based on the teiresias pattern-discovery algorithm," in *EICAR Annual Conference Proceedings*, pp. 1–15, 1999.

[20] S. Brin, R. Motiwani, and C. Silverstein, "Beyond market baskets: Generalizing association rules to correlations," *Data Mining and Knowledge Discovery*, vol. 2, pp. 39–68, 1998.

[21] R. Vaarandi, "A data clustering algorithm for mining patterns from event logs," in *IEEE IPOM'03 Proceedings*, 2003.

[22] S. Ma and J. Hellerstein, "Mining partially periodic event patterns with unknown periods," in *Proceedings of the 2001 International Conference on Data Engineering (ICDE'01)*, pp. 409–416, 2001.

[23] J. Laros and L. Ward, "Implementing scalable diskless clusters using the network file system (nfs)," in *LACSI 2003 Conference Proceedings*, 2003.